Consensus Across Continents

Anonymous Author(s)

Abstract

Distributing data storage systems across geographic areas provides resilience to catastrophic failure and localizes user accesses, decreasing latency and improving throughput. However, as network distance increases, the impact of failure modes such as partitions and communication variability pose challenges to coordination that impair strong consistency. As a result, geo-distributing systems implies a system size larger than a handful of replicas, requiring an extension of current consensus algorithms. While some current systems are sufficiently large to protect against failures, they are also rigid, specialized for application-specific access patterns that do not generalize well. In order to balance consistency and performance in a multi-region context, geo -distributed consensus must be flexible, adapting to changing network conditions and user behavior.

In this paper we introduce Alia, a hierarchical consensus protocol that is designed for agility and serves as a framework to create large, strongly-consistent, and adaptable geo-replicated consensus groups. Alia splits coordination responsibility across two tiers: a root quorum responsible for moving the system through reconfiguration safely, and subquorums which manage direct access to the data. Subquorums intersect with the root quorum using a novel method, delegated voting, which ensures that all replicas participate in both consensus tiers and provide transparent, linearizable guarantees across the entire system. This design ensures Alia can optimize throughput and availability by flexibly changing its configuration in real time to meet demand without sacrificing consistency.

CCS Concepts • **Information systems → Remote replication**; **Distributed storage**;

Keywords hierarchical consensus, geographic replication, delegated voting

SOSP 2019, October 27–30, 2019, Huntsville, Ontario, Canada

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference format:
Anonymous Author(s). 2019. Consensus Across Continents. In *Proceedings of SOSP 2019: ACM Symposium on Operating Systems Principles, Huntsville, Ontario, Canada, October 27–30, 2019 (SOSP 2019),* 10 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The availability of cloud services that span the globe has made it easier than ever before to deploy geographically distributed data systems which replicate over continents and oceans. These types of systems increase local performance by minimizing network distance between users and replicas and provide the opportunity for data recovery in the face of catastrophes such as floods or earthquakes. Moreover, the success of specialized, high-availability data systems [2] in maximizing throughput across the wide area has led to increased interest in geo-replication systems, particularly as more applications are being deployed with international audiences in mind. In order to generalize these systems, however, stronger consistency semantics are required to allow developers to reason correctly about the underlying behavior of the system.

The most straightforward mechanism to guarantee strong consistency is to coordinate a group of replicas using distributed consensus, canonically represented by Paxos [12] and its performance-optimizing variants [3, 5, 10, 11]. Although some recent research has explored the problem of geo-distributed consensus, specifically considering consensus with high latency links [14, 16], the distributed consensus problem primarily considers the case of safety with either one or two fail-stop node failures. Geo-replication, however, implies scaling; services running around the globe require dozens if not hundreds of replicas, and introduces new failure modes such as network partitions, where sections of the system operate independently without fail-stop failure, and highly variable latency that lead to changing network conditions. In order to scale systems beyond a handful of replicas, current systems [6, 8, 9, 17] use Paxos as a component, which can be instantiated across multiple transactions, shards, or tablets in order to manage subsystems that only require coordination between a few primary nodes.

But Paxos alone is not enough; on the one hand we have consensus algorithms, which provide strong consistency for small quorums, but result in communication bottlenecks at scale. On the other hand we have real world systems, which can be engineered to scale consistency, but that rely on expensive hardware and complex, rigid designs, and which cannot be made generally applicable.

What is missing is the middle ground, a general framework for geo-distributed systems that not only provides the highest possible throughput, but also the strongest consistency semantics in a network environment prone to correlated failures, partitions, and variable latency. Such a framework should provide:

- true horizontal scaling
- the ability to localize and optimize accesses to a variety of user patterns
- resilience beyond single node failure; e.g. in the face of partitions, but no opportunity for partial or ambiguous failure

It should moreover achieve these requirements with a single consensus layer, to facilitate implementation across a range of geographically distributed contexts, enable dynamic reconfiguration, and ensure straightforward system maintenance.

This paper introduces the first such framework, *hierarchical consensus*, a leader-oriented protocol that maintains multiple subquorums, each of which elects a leader to coordinate decisions. Hierarchical consensus coordinates all managed processes by organizing them into a tier of quorums such that parent quorums manage the *decision space* and leaf quorums manage *access ordering*. Each tier implements quorums that make decisions about their respective operations. To illustrate the fault tolerance, recovery properties, strong consistency guarantees, and throughput possible using the hierarchical consensus framework, we present Alia, the first distributed consensus protocol able to scale across hundreds of replicas around the world.

We begin by reviewing the problem of consistency across the wide area in Section 2. We present the high level details of the hierarchical consensus algorithm in Section 3, discussing the operations of the root quorum including delegated voting and epoch transitions. In Section 4 we delve further into the operations of subquorums and transactions. In Sections 5 and 6, we demonstrate the safety and consistency guarantees of hierarchical consensus, and present the experimental results of the Alia system in Section 7.

2 Background

We consider geographically replicated systems, which by their very nature scale to hundreds or thousands of processes working in concert to fulfill a myriad of client accesses from a wide area. The literature for such systems falls broadly into two categories; (1) new or optimized consensus algorithms designed to handle replication and failure in the wide area [3, 16], and (2) case studies of truly large-scale systems that implement distributed databases or other far-reaching applications [6, 8, 17]. In this section, we will describe the algorithmic foundations for strong consistency and the systems that have been built upon them, both of which provide an intuition for the design of Alia, a transparent, flexible framework that can offer strong consistency, horizontal scaling, and fault tolerance.

Modern cloud-scale applications developers are faced with geographic replication challenges routinely, but their options are limited to cloud service provider solutions [6] and theoretical works that do not consider scale [1, 9, 10, 13, 16]. Consider the trade-offs involved in designing a geographic replication system that must support a million users. One approach might privilege partition tolerance to ensure the operations can continue even if large portions of the system are not able to communicate. Another may allocate decision-making such that a few node failures will have no impact on the overall state of the system. Is it more important for decision-making to be efficient, requiring as few round-trips as possible, or for different parts of the system to be able to make decisions simultaneously? There are many questions which must be addressed; what is the best mechanism for adding and removing nodes? Will many objects be accessed in different locations at different times? How routinely will transactions require transatlantic commit? These decisions require a balance between performance and throughput, between recovery and fault tolerance, and not to mention day-to-day systems administration.

2.1 Geo-Distributed Consensus

Geographic replication has two problems, the first of which is high and variable latency between links, which makes throughput slow between replicas across the wide area. The second problem is a new type of failure, not of individual nodes, but partitions, wherein two parts of a network are otherwise functional but unable to communicate. Thus from the developer perspective, the goals are to minimize latency and to ensure that quorums will not be disrupted by partitions.

To provide fault tolerance and strong consistency, we rely on quorum-based algorithms that coordinate different processes on different machines. Largely these are all based on Paxos [11], which has theoretically determined that the safest way to coordinate across nodes is a two-phase approach—a leader nominates itself, the rest of the nodes accept the leader, the leader proposes values, and that slot in the log is committed. This style of consensus creates a single log of operations that exists on all machines, and when it is applied in the same order, all replicas arrive at the same state. From a performance perspective, the two-phase approach suffers from latency variability, and doesn't provide good throughput.

As far as we are aware, only three Paxos variants—Raft, Mencius, and ePaxos —have been used over the wide area. Raft attempts to eliminate the propose phase by pre-electing a leader responsible for the slots in the log. Some other mechanism is then used to determine if the leader has failed, and to elect a new leader. Along with command aggregation, this optimization reduces the number of messages that have to be sent across the wide area. This is good news for clients

collocated with the leader, since they can expect the best possible performance. Raft is thus well-suited in a geo-replicated context for a primary backup replica scenario, particularly across local availability zones, and for catastrophic failure recovery.

However, Raft is not ideal for clients that are geographically distant from the leader. An alternative option is Mencius, which assigns leadership in round robin fashion so that there is a known, pre-elected leader for every slot. This increases the likelihood that the client will be collocated with the leader for the best performance in the local area. Thus, Raft and Mencius both improve performance for certain access problems, such as the case where clients are predominately located in a single portion of the consensus area, or are roughly evenly distributed across a small consensus area.

Like Mencius, ePaxos uses a multi-leader approach, but optimistically uses thriftiness and a fast path to make commits more quickly on behalf of the client. This happens at the cost of a potential slow path during execution after the access is completed, which requires even more coordination between replicas. These techniques give the best possible throughput when clients are uniformly distributed across all replicas, and can even avoid partial partitions by repairing logs with multi-hop consensus. As conflict increases however, their advantages diminish.

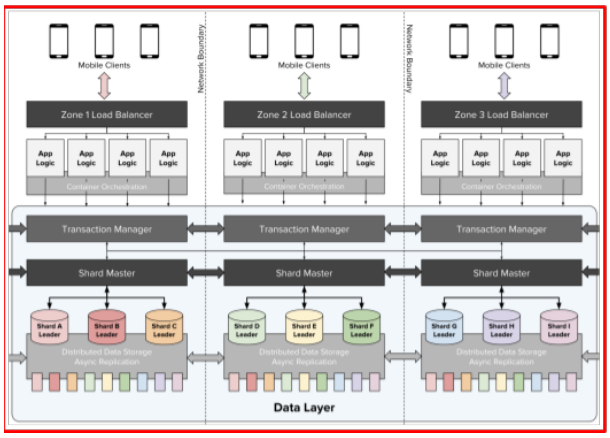
Though each of these three algorithms is useful in different contexts in the wide area, none scale. Consensus algorithms consider only single node failure and therefore only describe their behavior in the context of small quorums—three replicas allowing for one failure, or five replicas allowing for two to fail. As quorum size increases, the amount of coordination also increases, and so too does the amount of susceptibility to variable latency and partitions.

2.2 Engineering Solutions

In the previous section, we considered consensus in the wide area for strong consistency. For geographically distributed systems in the wild, the chief practical consideration is high throughout. This is achieved by partitioning the namespace into tablets of related objects that are managed together, and by placing those tablets in geographic proximity to their users. Early systems sacrificed availability for the sake of consistency, but today's strongly consistent systems use a complex architecture that depends on high performance data center hardware to make specific guarantees.

An architecture for a general geo-distributed system is shown in figure 1. A lockserver such as etcd [15] or Chubby [4] assigns tablets to either individual replicas or to small quorums. Accesses go through these quorums and are written to a distributed data store or a file system such as Collossus [7] or Ceph [18]. The replicas or quorums routinely renew their leases with the lockserver, but if they die or the lease expires, the tablet can be reassigned to another replica that can read

Figure 1. Example of a general geo-distributed system architecture



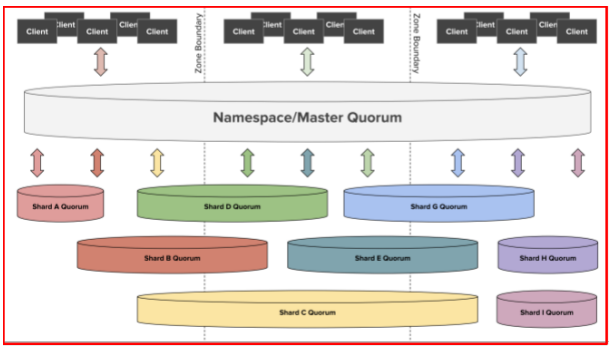
the data off the geo-distributed file system. For cross-tablet accesses, the state of the art is to use snapshot isolation with extremely precise timestamps or vector clocks. The benefit of this architecture is that the primary workload is handled by small quorums of the type described in the previous section. Moreover, the system is extremely robust to node failures, meaning they provide insurance against data loss.

Together, the three primary components of such systems—the name service, lock service, and distributed file system—achieve the foremost requirement: high performance for many simultaneous users. However, these system components are not safer merely because they operate independently from the rest of the system; a data store is not less susceptible to latency or more resilient to failure by virtue of being engineered separately from the lock server or name server. Their disjunction does not furnish any particular advantage; on the contrary, it introduces several new challenges.

First, the consistency properties are handled at multiple layers, adding complexity that makes it very difficult to reason about system-level consistency in a meaningful way. The lockserver can become a bottleneck; the primary replica, responsible for ordering accesses, may become imbalanced; and finally, the distributed data layer may have a completely difference consistency semantic. This means that such systems must rely on regular human intervention and coordination for administration and maintenance.

Moreover, the separation of decision-making across the components of these systems makes them extremely rigid; depending, for instance, on coarse epochs and expensive hardware to provide safety, and assuming, rather optimistically, that tablets will be accessed only inside a single region. Optimization for geographic accesses is thus necessarily very naive; executed via applications that store, as an example, the entirety of a user's inbox in a single location in spite of

Figure 2. A simplified architecture with a single consensus layer



having more than enough data on usage patterns to inform a more nuanced, even adaptive, storage strategy.

While these systems work, they are fundamentally limited by their design. What they really need is a mechanism for achieving the same high throughput and strong consistency without incurring the expenses of human coordination and the loss of system-wide interpretability. Most importantly, they should also provide the flexibility of the algorithmic approaches, endowing engineered systems with the opportunity to adapt to changing conditions, and untethering them from reliance on premium hardware. An example of a system designed using a single consensus layer is shown in figure 2.

2.3 A Framework for Agile Systems

Alia takes a different approach to implementing geo-distributed systems, focusing on a system's ability to be *flexible*. Flexibility ensures that the system can balance requirements for throughput and availability while still maintaining the strongest possible consistency semantics. To achieve this, Alia is based on three primary design requirements that inform the rest of the framework.

Requirement 1: Systems should be as fluid as the information they contain. Many systems are optimistic, they assume that conflict is rare and that objects are accessed in standard patterns that change. In our experience, both people and information flows freely therefore a system must accommodate organic and shifting usage patterns; for example a set of objects may primarily be accessed only in daylight, requiring the system to adapt by moving the coordinating replicas to the locales currently in working hours. To accommodate this requirement, Alia is designed to regularly and safely transition through reconfigurations called epoch changes, reallocating replicas into subquorums to manage specific partitions of the namespace. Epoch changes are *fuzzy* to ensure that reconfiguration does not need to be synchronous and hand-offs are optimized through anti-entropy replication of data.

Requirement 2: No partial failures. A system's size should be its advantage – allowing increased throughput with linear scaling, and better placement to optimize accesses. Often, however, a system's size increases its complexity and it's susceptibility to unique failures such as correlated cascading failure.

Alia is designed with a single process model – the same process participating in the root quorum also handles messages for the subquorum(s) the process has been assigned to. This model ensures that if a replica fails it cannot participate in some decision making, such as configuration, but not others, such as accesses. This requirement also allows us to more easily tackle complex failures; such as using a nuclear option (discussed in 5.3) to ensure progress even with a worst-case failure of delegates, or ensuring that leases are either respected or replaced for whole subquorums that fall out of communication.

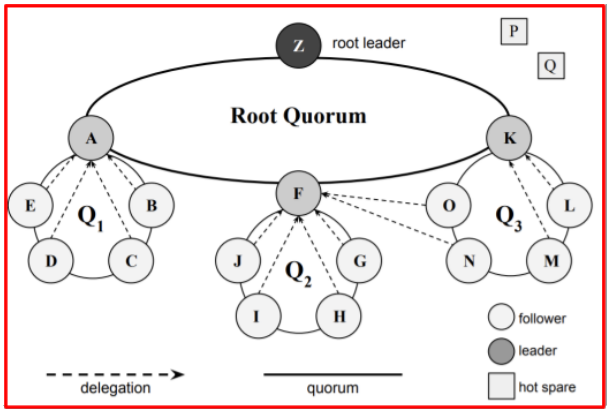
Requirement 3: Consistency semantics must be transparent and interpretable. As privacy and security become increasingly important requirements of distributed systems, consistency is no longer about ensuring that your boss cannot see your Spring Break pictures on a social network wall. Instead, consistency is about ensuring that the correct operations are being executed on the correct replicas and that data can be audited to discover its exact placement. Alia ensures that there is an intersection between subquorums where data accesses are taking place and the root quorum where configuration and namespace partitions are occurring. This intersection is optimized by delegated voting to ensure that the root quorum can make progress and remain fluid. The intersection also guarantees that a complete, externalizable log of events for the global system can be exported on demand.

3 Hierarchical Consensus

Hierarchical consensus is an implementation and extension of Vertical Paxos [13] that organizes replicas into two tiers of quorums, each responsible for fundamentally different decisions, as shown in Figure 3. The lower tier consists of multiple independent subquorums, each committing operations to local shared logs. The upper, root quorum, consists of subquorum peers, usually their leaders, delegated to represent the subquorum and hot spares in root elections and commits. Hierarchical consensus's main function is to export a linearizable abstraction of shared accesses to some underlying substrate, such as a distributed object store or file system. We assume that nodes hosting object stores, applications, and HC are frequently co-located across the wide area.

The root quorum's primary responsibilities are mapping replicas to individual subquorums and mapping subquorums

Figure 3. The root quorum coordinates all replicas in the system including hot spares, though active participation is only by delegated representatives of subquorums, which do not necessarily have to be leaders of the subquorum, though this is most typical. Subquorums are configured by root quorum decisions which determine epochs of operation. Each subquorum handles accesses to its own independent portion of the namespace.



to tags within the namespace. Each such map defines a distinct epoch, e_x, a monotonically increasing representation of the term of the configuration of subquorums and tags, Q_x. The root quorum is a consensus group consisting of subquorum leaders. Somewhat like subquorums, the membership of the root quorum is not defined by the quorum itself, but in this case by leader election or peer delegations in the lower tier. While the root quorum is composed of all replicas in the system, only this subset of replicas actively participates in root quorum decision making, in the common case.

The root quorum partitions (shards) the namespace across multiple subquorums, each with a disjoint portion as its scope. The namespace is decomposed into a set of tags, T where each tag t_i is a disjoint subset of the namespace.

Tags are mapped to subquorums in each epoch, $Q_x \mapsto T_x$ such that $\forall t \in T_x \; \exists! q_{i,x} \mapsto t$. The intent of subquorum localization is ensure that the *domain* of a client, the portion of the namespace it accesses, is entirely within the scope of a local, or nearby, subquorum. To the extent that this is true across the entire system, each client interacts with only one subquorum, and subquorums do not interact at all during execution of a single epoch. This *siloing* of client accesses simplifies implementation of strong consistency guarantees and allows better performance at the cost of restricting multi-object transactions. We use agility to attempt to get this, but allow multi-object transactions.

3.1 Delegated Voting

From a logical perspective, the root quorum's membership is the set of all system replicas, at all times. However, running consensus elections across large systems is inefficient in the best of cases, and prohibitively slow in a geo-replicated environment. Root quorum decision-making is kept tractable by having replicas *delegate* their votes, usually to their local leaders, for a finite duration of epochs. With leader delegation, the root membership effectively consists of the set of subquorum leaders. Each leader votes with a count describing its own and peer votes from its subquorum and from hot spares that have delegated to it. A quorum leader is elected to indefinitely assign log entries to slots (access operations for subquorums, epoch configurations for the root quorum). If the leader fails, then so long as the quorum has enough online peers, they can elect a new leader. When a failed leader comes back online, it rejoins the quorum as a follower. The larger the size of the quorum, the more failures it is able to tolerate.

Delegation ensures that root quorum membership is always the entire system and remains unchanged over subquorum leader elections and even reconfiguration. Delegation is essentially a way to optimistically shortcut contacting every replica for each decision. Subquorum repartitioning merely implies that a given replica's vote might need to be delegated to a different leader. To ensure that delegation happens correctly and without requiring coordination, we simply allow a replica to directly designate another replica as its delegate until some future epoch is reached. Replicas may only delegate their vote once per epoch and replicas are not required to delegate their vote. To simplify this process, during configuration of subquorums by the root quorum, the root leader provides delegate hints, e.g. those replicas that have been stable members of the root quorum without partitions. When replicas receive their configuration they can use these hints to delegate their vote to the closest nearby delegate if not already delegated for the epoch. If no hints are provided, then replica followers generally delegate their vote to the term 1 leader and hot spares to the closest subquorum leader.

Delegation does add one complication: the root quorum leader must know all vote delegations to request votes when committing epoch changes. We deal with this issue by simplifying our protocol. Instead of sending vote requests just to subquorum leaders, **the root quorum leader sends vote requests to all system replicas.** This is true even for *hot spares*, which are not currently in any subquorum. Delegates reply with the unique ids of the replicas they represent so

that root consensus decisions are still made using a majority of all system replicas.

This is correct because vote requests now reach all replicas, and because replicas whose votes have been delegated merely ignore the request. We argue that it is also efficient, as a commit's efficiency depends only on receipt of a majority of the votes. Large consensus groups are generally slow, not just because of communication latency, but because large groups in a heterogeneous setting are more likely to include replicas on very slow hosts or networks. In the usual case for our protocol, the root leader still only needs to wait for votes from the subquorum leaders. Leaders are generally those that respond more quickly to timeouts, so the speed of root quorum operations is unchanged.

3.2 Epoch Transitions

Every epoch represents a new configuration of the system as designated by the root leader. Efficient reconfiguration ensures that the system is both dynamic, responding both to failures and changing usage patterns, and minimizes coordination by colocating related objects. An epoch change is initiated by the root leader in response to one of several events, including:

- a namespace repartition request from a subquorum leader
- notification of join requests by new replicas
- notification of failed replicas
- changing network conditions that suggest re-assignment of replicas
- manual reconfigurations, e.g. to localize data

The root leader transitions to a new epoch through the normal commit phase in the root quorum. The command proposed by the leader is an enumeration of the new subquorum partition, namespace partition, and assignment of namespace portions to specific subquorums. The announcement may also include initial leaders for each subquorum, with the usual rules for leader election applying otherwise, or if the assigned leader is unresponsive. Upon commit, the operation serves as an *announcement* to subquorum leaders. Subquorum leaders repeat the announcement locally, disseminating full knowledge of the new system configuration, and eventually transition to the new epoch by committing an `epoch-change` operation locally.

The epoch change is lightweight for subquorums that are not directly affected by the overarching reconfiguration. If a subquorum is being changed or dissolved, however, the *epoch-change* commitment becomes a tombstone written to the logs of all local replicas. No further operations will be committed by that version of the subgroup, and the local shared log is archived and then truncated. Truncation is necessary to guarantee a consistent view of the log within a subquorum, as peers may have been part of different subquorums, and thus have different logs, during the last epoch. Replicas then begin participating in their new subquorum instantiation. In the common case where a subquorum's membership remains unchanged across the transition, an `epoch-change` may still require additional mechanism because of changes in namespace responsibility.

3.3 Elastic Quorum Membership

In principle, adding or removing a node from the system simply involves a root quorum decision that changes the epoch and reconfigures the system. However, unlike other configuration changes, elastic membership has implications for safety and fault tolerance, which we will discuss in this section.

4 Epoch Operations

4.1 Remote Accesses

4.2 Transactions

5 Safety

5.1 Expected Failure

5.2 Assassination

5.3 Nuclear Option

6 Consistency and Adaptability

6.1 Exporting a Global Log

6.2 Adapting Consistency

7 Performance Evaluation

7.1 Experimental Setup

As shown in figures 4 and 5, our implementations of ePaxos and Raft are correct.

7.2 Wide-Area Throughput

As we see in figure 6, as nodes are added to the system, we get true horizontal scaling.

Figure 4. Latency of Raft and ePaxos with 3, 5, and 7 replica quorums

Plot shows performance of 3, 5, 7 replica quorums of Raft and ePaxos.

Axes show different conflict levels vs. quorum sizes.

Purpose is to give the reader confidence that our versions of Raft and ePaxos are correct.

Similar to the below from ePaxos paper:

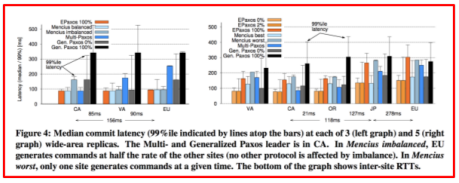


Figure 4: Median commit latency (99%ile indicated by lines atop the bars) at each of 3 (left graph) and 5 (right graph) wide-area replicas. The Multi- and Generalized Paxos leader is in CA. In *Mencius imbalanced*, EU generates commands at half the rate of the other sites (no other protocol is affected by imbalance). In *Mencius worst*, only one site generates commands at a given time. The bottom of the graph shows inter-site RTTs.

Figure 5. Throughput of Raft and ePaxos with 3, 5, and 7 replica quorums

Plot shows performance of 3, 5, 7 replica quorums of Raft and ePaxos.

Axes show different conflict levels vs. quorum sizes.

Purpose is to give the reader confidence that our versions of Raft and ePaxos are correct.

Similar to the below from ePaxos:

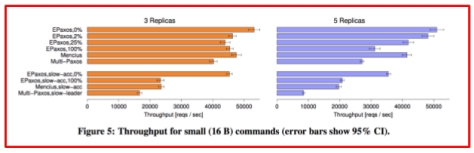


Figure 5: Throughput for small (16 B) commands (error bars show 95% CI).

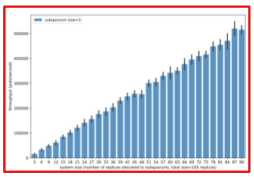
Figure 6. Linear scaling of throughput with Alia

Plot shows throughput as Alia scales to 144 replicas across 16 regions.

Show how big we can make this and how does the throughput scale.

Plot shows linear scaling as Alia scales.

Similar to this plot from the dissertation:



7.3 Commit Latency and Conflicts

As shown in figure 7, Alia is has comparable performance to ePaxos in a zero-conflict context.

Figure 7. Latency of Alia with respect to ePaxos

Plot showing many region alia vs epaxos

vs. latency and vs conflicts

(should be equivalent to 0-conflict epaxos)

Shows a comparison of what happens when there is no conflict in the wide area, e.g. 3X3 alia with Raft quorums in VA, Montreal, CA

Figure 8. Alia is fault tolerant in the face of subquorum failure

Time series plot shows that even with faults, Alia is still robust.

Create partitions or node failures, show how Alia reconfigures.

Have 3 subquorums running, have one fail; throughput tanks, then Alia reconfigures and watch other 2 assume the load of the failed subquorum

Show flexibility by showing variable commit latency depending on type of object, e.g. objects are only accessed with a latency subject to the replication links they are associated with and no more.

7.4 Fault Tolerance and Adaptability

1. Show what happens when subquorums fail (epoch change recovery), shown in figure 8.

2. Show sawtooth in figure 9: as accesses move to different regions

3. Stretch: show assassination recovery?

7.5 Anti-Entropy Optimization

As we can see in figure 10, anti-entropy improves hand-offs and durability.

Figure 9. Alia repairs client accesses as they migrate to different regions

Time series plot shows a sawtooth graph with Alia repairing client accesses

This is about flexibility; illustrates how the system recovers as access patterns move around the globe according to daylight/work hours.

Start with quorum in VA with all clients in VA, one by one clients move to OH, then Alia moves subquorum to OH, see throughput spike, then clients move to OR, etc

Figure 10. Anti-entropy improves hand-offs and durability

Plot shows performance improvements with anti-entropy optimizations.

Shows how anti-entropy improves hand-offs and durability

Show percent replication over time of an object - what does that mean for the number of objects that need to be handed off?

Perhaps drawn with a function rather than with empirical results?

Figure 11. Best case scenarios for Alia

Plot shows venn-style diagram of optimal case for Alia.

Perhaps drawn with a function rather than with empirical results?

Similar to this from Fawn:

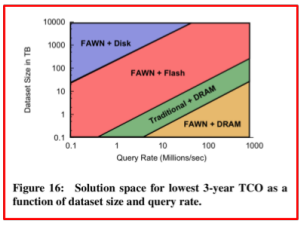


Figure 16: Solution space for lowest 3-year TCO as a function of dataset size and query rate.

8 Related Work

Figure 11 shows the conditions for which Alia is most optimal.

9 Conclusion

Where is the data? ITAR and GDPR and other policies are going to become increasingly influential on the technical landscape.

References

[1] Ailidani Ailijiang, Aleksey Charapko, Murat Demirbas, and Tevfik Kosar. [n. d.]. WPaxos: Ruling the Archipelago with Fast Consensus. ([n. d.]).

[2] Jason Baker, Chris Bond, James C. Corbett, J. J. Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. [n. d.]. Megastore: Providing Scalable, Highly Available Storage for Interactive Services.. In *CIDR* (2011), Vol. 11. 223–234. http://pages.cs.wisc.edu/~akella/CS838/F12/838-CloudPapers/Megastore.pdf

[3] Martin Biely, Zoran Milosevic, Nuno Santos, and Andre Schiper. [n. d.]. S-Paxos: Offloading the Leader for High Throughput State Machine Replication. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium On* (2012). IEEE, 111–120. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6424845

[4] Mike Burrows. 2006. The Chubby Lock Service for Loosely-Coupled Distributed Systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. USENIX Association, 335–350. http://dl.acm.org/citation.cfm?id=1298487

[5] Lásaro Jonas Camargos, Rodrigo Malta Schmidt, and Fernando Pedone. [n. d.]. Multicoordinated Paxos. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing* (2007). ACM, 316–317. http://dl.acm.org/citation.cfm?id=1281150

[6] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. [n. d.]. Spanner: Google's Globally Distributed Database. 31, 3 ([n. d.]), 8. http://dl.acm.org/citation.cfm?id=2491245

[7] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. [n. d.]. The Google File System. In *ACM SIGOPS Operating Systems Review* (2003), Vol. 37. ACM, 29–43. http://dl.acm.org/citation.cfm?id=945450

[8] Lisa Glendenning, Ivan Beschastnikh, Arvind Krishnamurthy, and Thomas Anderson. [n. d.]. Scalable Consistency in Scatter. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011). ACM, 15–28. http://dl.acm.org/citation.cfm?id=2043559

[9] Tim Kraska, Gene Pang, Michael J. Franklin, Samuel Madden, and Alan Fekete. [n. d.]. MDCC: Multi-Data Center Consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems* (2013). ACM, 113–126.

[10] Leslie Lamport. [n. d.]. Fast Paxos. 19, 2 ([n. d.]), 79–103. http://link.springer.com/article/10.1007/s00446-006-0005-x

[11] Leslie Lamport. [n. d.]. Generalized Consensus and Paxos. ([n. d.]). http://zoo.cs.yale.edu/classes/cs426/2012/lab/bib/lamport05generalized.pdf

[12] Leslie Lamport. [n. d.]. Paxos Made Simple. 32, 4 ([n. d.]), 18–25. http://www.cs.utexas.edu/users/lorenzo/corsi/cs380d/past/03F/notes/paxos-simple.pdf

[13] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. [n. d.]. Vertical Paxos and Primary-Backup Replication. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing* (2009). ACM, 312–313.

[14] Yanhua Mao, Flavio Paiva Junqueira, and Keith Marzullo. [n. d.]. Mencius: Building Efficient Replicated State Machines for WANs. In *OSDI* (2008), Vol. 8. 369–384. https://www.usenix.org/legacy/event/osdi08/tech/full_papers/mao/mao_html/

[15] Blake Mizerany, Qin Yicheng, and Li Xiang. [n. d.]. Etcd: Package Raft. ([n. d.]). https://godoc.org/github.com/coreos/etcd/raft

[16] Iulian Moraru, David G. Andersen, and Michael Kaminsky. [n. d.]. There Is More Consensus in Egalitarian Parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013). ACM, 358–372. http://dl.acm.org/citation.cfm?id=2517350

[17] Alexander Thomson and Daniel J. Abadi. [n. d.]. CalvinFS: Consistent Wan Replication and Scalable Metadata Management for Distributed File Systems. In *13th USENIX Conference on File and Storage Technologies (FAST 15)* (2015). 1–14. https://www.usenix.org/node/188413

[18] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell DE Long, and Carlos Maltzahn. 2006. Ceph: A Scalable, High-Performance Distributed File System. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. USENIX Association, 307–320.